02

# Lidando com Formulários e o Cadastro de Produtos

Temos um sistema capaz de listar produtos cadastrados, vamos agora incrementá-lo com a capacidade de cadastrar novos produtos.

Para realizar o cadastro, precisamos inicialmente de um formulário que receba os dados necessários para se criar um novo produto.

```
<form action="/Produto/Adiciona">
  <label for="nome">Nome:</label>
  <input id="nome" name="nome" />

  <label for="preco">Preço:</label>
  <input id="preco" name="preco" />

  <label for="quantidade">Quantidade:</label>
  <input id="quantidade" name="quantidade" />

  <label for="descricao">Descrição:</label>
  <input id="descricao" name="descricao" />

  <label for="categoria">Categoria:</label>
  <input id="categoria" name="categoriaId" />

  <input type="submit" />
</form>
```

Mas como mencionado no começo do curso, esse formulário faz parte da camada de visualização do MVC, logo precisamos criar um método no controller que, por enquanto, não faz nada a não ser redirecionar para a view com o formulário. Vamos então adicionar um método chamado `Form` dentro do `ProdutoController` .

```
public ActionResult Form()
{
    return View();
}
```

Podemos ver o formulário gerado acessando o endereço: `/Produto/Form` com o navegador.


## Recebendo as informações do formulário na action

Vamos agora criar um novo método no `ProdutoController` que será responsável por receber as informações que foram enviadas pelo formulário. Como no formulário colocamos a action `/Produto/Adiciona` , então os dados do formulário serão enviados para o método `Adiciona` do `ProdutoController` :

```
public ActionResult Adiciona()
{
    // implementação da action
}
```

Dentro dessa action, para utilizarmos as informações que foram enviadas pelo navegador, precisamos declarar parâmetros no método que possuem o mesmo nome do campo de texto no html (atributo `name` da tag `input`).

```
public ActionResult Adiciona(String nome, float preco, int quantidade, String descricao, int categor
```

Dentro desse método, precisamos instanciar um novo produto com as informações recebidas:

```
Produto produto = new Produto() {
    Nome = nome,
    Preco = preco,
    Quantidade = quantidade,
    Descricao = descricao,
    CategoriaId = categoriaId
};
```

E, por fim, utilizaremos o DAO para adicionar o produto no banco de dados:

```
ProdutosDAO dao = new ProdutosDAO();
dao.Adiciona(produto);
```

O código final de nossa action fica da seguinte forma:

```
public ActionResult Adiciona(String nome, float preco, int quantidade, String descricao, int categor
{
    Produto produto = new Produto() {
        Nome = nome,
        Preco = preco,
        Quantidade = quantidade,
        Descricao = descricao,
        CategoriaId = categoriaId
    };
    ProdutosDAO dao = new ProdutosDAO();
    dao.Adiciona(produto);
    return View();
}
```

Repare que os parâmetros enviados pelo formulário são automaticamente convertidos para os tipos certos. Essa conversão é realizada pelo ASP.NET MVC.

## Envio de dados para uma action através de model binders

Em nossa solução para o problema de cadastro de produtos, foi necessário instanciar manualmente a classe `Produto` utilizando os parâmetros da requisição, porém seria muito mais interessante se o método recebesse como argumento uma instância de `Produto` populada com os parâmetros da requisição.

```
public ActionResult Adiciona(Produto produto)
{
    ProdutosDAO dao = new ProdutosDAO();
    dao.Adiciona(produto);
    return View();
}
```

Para que esse código funcione, os campos do formulário devem ter nome da forma `nomeDaVariavel.NomeDoAtributo`, por exemplo, se quisermos que o atributo `Nome` do produto seja preenchido automaticamente, devemos criar um formulário HTML que tenha um campo chamado `produto.Nome`: `<input name="produto.Nome" />`.

```
<form action="/Produto/Adiciona">
  <label for="nome">Nome:</label>
  <input id="nome" name="produto.Nome" />

  <label for="preco">Preço:</label>
  <input id="preco" name="produto.Preco" />

  <label for="quantidade">Quantidade:</label>
  <input id="quantidade" name="produto.Quantidade" />

  <label for="descricao">Descrição:</label>
  <input id="descricao" name="produto.Descricao" />

  <label for="categoria">Categoria:</label>
  <input id="categoria" name="produto.CategoriaId" />

  <input type="submit" />
</form>
```

A variável `produto` recebida pelo método da classe controladora é instanciada e preenchida por um componente do ASP.NET MVC conhecido como **Model Binder**.

Os model binders do ASP.NET MVC são componentes que convertem os parâmetros da requisição para objetos. Tanto no primeiro caso, em que recebíamos tipos simples, quanto no segundo caso, em que recebemos uma instância de produto, é o model binder que realiza a conversão de tipos.

## Formulários utilizando o Post

Ao enviarmos o formulário que desenvolvemos anteriormente, podemos ver que a url mostrada pelo navegador contem as informações preenchidas no formulário, isso acontece pois essa é a forma de envio de informações utilizada por padrão pelo navegador, o método de envio chamado **Get**.

O Get é geralmente utilizado quando desenvolvemos uma busca dentro da aplicação, pois como todas as informações do formulário de busca estão na url do navegador, podemos facilmente compartilhar os resultados com outras pessoas.

Mas em um cadastro de informações não queremos deixar todas as informações expostas na url, nesses casos, podemos utilizar um outro método de envio chamado post. Para utilizarmos o post como método de envio do formulário, precisamos apenas colocar o atributo `method` na declaração da tag form:

```
<form action="/Produto/Adiciona" method="post">
  <!-- código do formulário -->
</form>
```

Depois dessa modificação, se testarmos novamente o cadastro de produtos, o navegador mostrará a url `/Produto/Adiciona` sem as informações que foram preenchidas no formulário.

Vimos que podemos enviar tanto requisições do tipo Get quanto do tipo Post para a action `Adiciona` do `ProdutoController` que o cadastro continuará funcionando, porém essa action deveria aceitar apenas requisições do tipo post, pois não estamos implementando uma busca. Quando queremos limitar o tipo de requisição que uma action recebe para o tipo post, utilizamos uma anotação (attribute do C#) chamada `HttpPostAttribute` sobre a declaração do método:

```
[HttpPostAttribute]
public ActionResult Adiciona(Produto produto)
```

Como por convenção toda anotação do C# termina com o sufixo `Attribute`, podemos simplificar o código para:

```
[HttpPost]
public ActionResult Adiciona(Produto produto)
```

Da mesma forma que podemos utilizar o `HttpPostAttribute` para aceitar apenas requisições do tipo post, também temos o `HttpGetAttribute` para aceitar apenas requisições do tipo Get.

## Redirecionando para a lista de produtos

Após a execução do método `Adiciona`, o ASP.NET MVC tentará renderizar a view `Produto/Adiciona.cshtml`, porém queremos que o usuário seja redirecionado para a página de listagem de produtos ( `Index` ). Poderíamos repetir o mesmo código que fizemos no método `Index` aqui, mas isso não faz sentido.

Para contornar esse problema, utilizaremos um novo tipo de resultado definido na classe `Controller`, o `RedirectToAction`.

```
public ActionResult Adiciona(Produto produto)
{
    ProdutosDAO dao = new ProdutosDAO();
    dao.Adiciona(produto);
    return RedirectToAction("Index");
}
```

O `RedirectToAction` redirecionará o usuário para o método `Index` do controller atual ( `ProdutoController` ).

## Melhorando a usabilidade do formulário com o Combo Box

Repare que no formulário que criamos, o usuário precisa digitar o `id` de uma categoria existente no banco de dados.

```
<label for="categoria">Categoria:</label>
<input id="categoria" name="categoriaId" />
```

Mas o id não é uma informação que o usuário deveria conhecer. Seria mais interessante se pudéssemos selecionar o nome a partir de uma lista de categorias cadastradas.

Vamos então utilizar a tag html `select` para exibir os nomes das categorias existentes e enviar o valor do `Id` selecionado para o servidor

```
<label for="categoria">Categoria:</label>
<select id="categoria" name="produto.CategoriaId">
  @foreach(var categoria in ViewBag.Categorias)
  {
    <option value="@categoria.Id">@categoria.Nome</option>
  }
</select>
```

E agora modificar nosso formulário para utilizar a tag `select` :

```
<form action="/Produto/Adiciona">
  <label for="nome">Nome:</label>
  <input id="nome" name="produto.Nome" />

  <label for="preco">Preço:</label>
  <input id="preco" name="produto.Preco" />

  <label for="quantidade">Quantidade:</label>
  <input id="quantidade" name="produto.Quantidade" />

  <label for="descricao">Descrição:</label>
  <input id="descricao" name="produto.Descricao" />

  <label for="categoria">Categoria:</label>
  <select id="categoria" name="produto.CategoriaId">
    @foreach(var categoria in ViewBag.Categorias)
    {
      <option value="@categoria.Id">@categoria.Nome</option>
    }
  </select>

  <input type="submit" />
</form>
```

Agora precisamos fazer com que a variável `ViewBag.Categorias` contenha a lista de categorias. Para isso, vamos modificar o método `Form` do `ProdutoController` :

```
public ActionResult Form()
{
    CategoriasDAO dao = new CategoriasDAO();
    IList<CategoriaDoProduto> categorias = dao.Lista();
    ViewBag.Categorias = categorias;
```

```
        return View();
    }
```

Visualize seu formulário pela url `/Produto/Form` .